课程回顾

- 动态规划问题：钢条切割问题、矩阵链乘法最优括号化、多边形最佳三角剖分
- 动态规划原理：
  - 最优子结构
  - 注意事项（子问题独立性）
  - 重叠子问题
  - 重构最优解

备忘

- 动态规划：分析是自顶向下，实现是自底向上
- 可采用备忘（记忆）型版本：采用自顶向下实现，是一个记忆型递归算法
- 将子问题的解记录在一个表中：
  - 每个子问题的解对应一表项
  - 每个表项初值为一个特殊值，表示尚未填入
  - 在递归算法执行过程中第一次遇某子问题时，计算其解并填入表中，以后再遇此子问题时，将表中值简单地返回（不重复计算），截断递归

备忘 (续)

```
MEMOIZED_MATRIX_CHAIN(p)
n ← p.length - 1
let m[1..n, 1..n] be a new table
for i ← 1 to n do
    for j ← 1 to n do
        m[i, j] ← ∞
return LOOKUP_CHAIN(m, p, 1, n)
```

$\Theta(n^2)$

```
LOOKUP_CHAIN(m, p, i, j)
if m[i, j] < ∞
    return m[i, j]
if i = j
    m[i, j] ← 0
else for k ← i to j-1
        q ← LOOKUP_CHAIN(m, p, i, k)
            + LOOKUP_CHAIN(m, p, k+1, j) + p_{i-1} p_k p_j
        if q < m[i, j]
            m[i, j] ← q
return m[i, j]
```

共$\Theta(n^2)$项

计算每个表项需$O(n)$

算法总时间$O(n^3)$

备忘 (续)

- 若所有子问题须至少解一次，自底向上的动态规划时间常数因子较优（不需要递归开销，维护表的开销较小）
- 若子问题空间有些不需要计算，则备忘型递归具有只需计算需要的子问题的优点

动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- **最长公共子序列**
- 最优二叉搜索树
- 0-1背包

最长公共子序列LCS

■ 子序列：将给定序列中零个或多个元素去掉之后得到的结果

- 形式化定义：给定一个序列$X=<x_1, x_2, ..., x_m>$，另一个序列$Z=<z_1, z_2, ..., z_k>$满足如下条件时称为X的子序列：存在一个严格递增的X的下标序列$<i_1, i_2, ..., i_k>$，对所有$j=1, 2, ..., k$，满足$x_{i_j}=z_j$
- 例：X=<A, B, C, B, D, A, B>
  Z=<　B, C,　D,　B>为X的子序列，对应下标序列为<2, 3, 5, 7>

子序列不一定是由原序列连续元素构成的序列！

最长公共子序列LCS (续)

- 公共子序列 (common subsequence)：给定两个序列X和Y，如果Z既是X的子序列，也是Y的子序列，则称Z是X和Y的公共子序列
- 最长公共子序列问题 (longest-common-subsequence problem)：求两个序列公共子序列中最长的一个
- 求解两个给定序列的LCS
  1. 刻画LCS结构特征
  2. 递归解
  3. 计算LCS长度
  4. 构造LCS

最长公共子序列LCS (续)

1. 刻画LCS特征

- 穷举法：穷举X的所有子序列，检查是否也是Y的子序列。若$|X|=m$，则子序列共2^m个，穷举法为指数阶下界
- LCS具有最优子结构性质
  前缀：给定一个序列$X=\langle x_1, x_2, ..., x_m\rangle$，对$i=0, 1, ..., m$，定义$X$的第$i$前缀为$X_i=\langle x_1, x_2, ..., x_i\rangle$
- **定理15.1** 令$X=\langle x_1, x_2, ..., x_m\rangle$和$Y=\langle y_1, y_2, ..., y_n\rangle$为两个序列，$Z=\langle z_1, z_2, ..., z_k\rangle$为$X$和$Y$的任意LCS
  1. 若$x_m=y_n$，则$z_k=x_m=y_n$且Z_{k-1}是X_{m-1}和Y_{n-1}的一个LCS；
  2. 若$x_m\neq y_n$，则$z_k\neq x_m$意味着Z是X_{m-1}和Y的一个LCS；
  3. 若$x_m\neq y_n$，则$z_k\neq y_n$意味着Z是X和Y_{n-1}的一个LCS。

最长公共子序列LCS (续)

1. 刻画LCS特征

- **定理15.1** 证明（反证法）：

  1. (1) $z_k=x_m=y_n$：若$z_k \neq x_m$，则可将$x_m=y_n$追加到Z的末尾，得到X和Y的一个长度为k+1的公共子序列，矛盾！

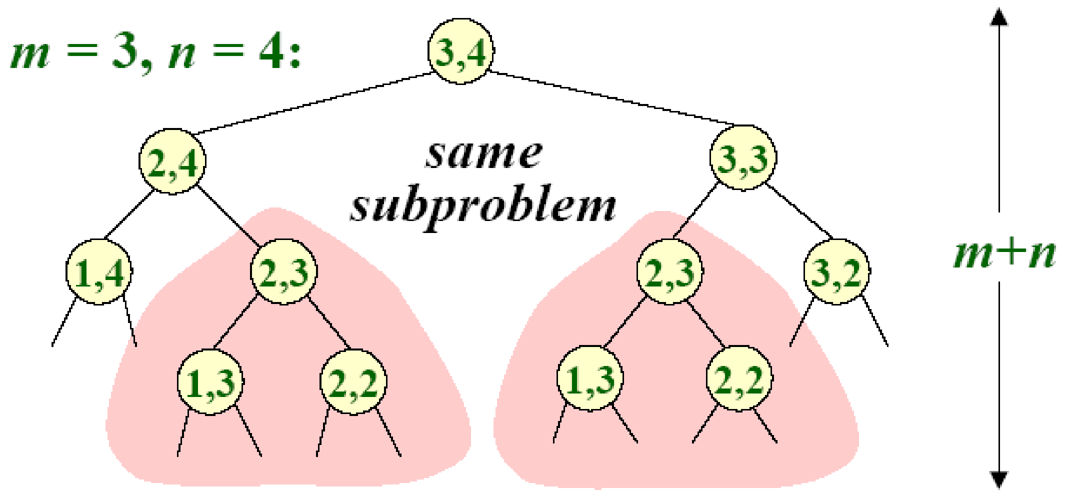
     (2) $Z_{k\text{-}1}$是$X_{m\text{-}1}$和$Y_{n\text{-}1}$的一个LCS：若$X_{m\text{-}1}$和$Y_{n\text{-}1}$存在长度大于k-1的公共子序列W，则可将$x_m=y_n$追加到W末尾，得到X和Y的一个长度大于k的公共子序列，矛盾！

  2. 因为$x_m \neq y_n$，所以$X_{m\text{-}1}$和Y的LCS与X和Y的LCS相同。若存在$X_{m\text{-}1}$和Y长度大于k的公共子序列W，则W也是X和Y的公共子序列，长度大于k，矛盾！

  3. 与2对称

最长公共子序列LCS (续)

1. 刻画LCS特征
   - 最优子结构性质：两个序列的一个LCS包含了两个序列的前缀子序列的一个LCS
     - 蕴含的选择：当$x_m \neq y_n$时，我们事先并不知道Z的长度，只能在X_{m-1}和Y的LCS以及在X和Y_{n-1}的LCS中取最大者
   - 重叠子问题性质



最长公共子序列LCS (续)

2. 递归解

- 由定理15.1：求$X=<x_1, x_2, ..., x_m>$和$Y=<y_1, y_2, ..., y_n>$的一个LCS时，需求解一个或两个子问题：
  - 若$x_m=y_n$，则求解X_{m-1}和Y_{n-1}的一个LCS，将$x_m=y_n$追加到这个LCS的末尾
  - 若$x_m \neq y_n$，(1) 求解X_{m-1}和Y的LCS；(2) 求解X和Y_{n-1}的LCS。求两者长度最大者

最长公共子序列LCS (续)

2. 递归解

- $c[i, j]$：X_i和Y_j的LCS长度（$0 \le i \le m$, $0 \le j \le n$）

$$c[i,j] = \begin{cases} 0, & i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1, & i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]), & i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

- 限定了需求解哪些子问题，并非所有子问题都要求解

最长公共子序列LCS (续)

3. 计算LCS长度

- 不同子问题个数：$\Theta(mn)$
- 输入：序列X=<$x_1, x_2, \ldots, x_m$>和Y=<$y_1, y_2, \ldots, y_n$>
- 输出：$c[0..m, 0..n]$：按行主次序计算LCS长度
  $b[1..m, 1..n]$：辅助构造最优解子序列

$$b[i,j] = \begin{cases} \nwarrow, & c[i,j] = c[i-1,j-1]+1, \\ \uparrow, & c[i,j] = c[i-1,j], \\ \leftarrow, & c[i,j] = c[i,j-1]. \end{cases}$$

- 构造解时，从$b[m, n]$出发，根据箭头方向上溯至i=0或j=0为止，当$b[i, j]$包含“↖”时打印出x_i即可

最长公共子序列LCS (续)

```
LCS_LENGTH(X, Y)
    m ← X.length
    n ← Y.length
    let b[1..m, 1..n] and c[0..m, 0..n] be new tables
    for i ← 1 to m do
        c[i, 0] ← 0
    for j ← 0 to n do
        c[0, j] ← 0
    for i ← 1 to m do        // 依次考虑X_1, X_2, ..., X_m的前缀子列
        for j ← 1 to n do    // 依次考虑Y_1, Y_2, ..., Y_n的前缀子列
            if x_i = y_j             // 两个前缀子列的最后一位相同
                c[i, j] ← c[i-1, j-1] + 1
                b[i, j] ← "↖"
            elseif c[i-1, j] ≥ c[i, j-1] // 两个前缀子列的最后一位不同
                c[i, j] ← c[i-1, j]
                b[i, j] ← "↑"
            else c[i, j] ← c[i, j-1]
                b[i, j] ← "←"
    return c and b
```

$\Theta(mn)$

最长公共子序列LCS (续)

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	←1	↖ 1
2	B	0	↖ 1	←1	←1	↑ 1	↖ 2	←2
3	C	0	↑ 1	↑ 1	↖ 2	←2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	←3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

X=<A, B, C, B, D, A, B>

Y=<B, D, C, A, B, A>

最长公共子序列LCS (续)

4. 构造LCS

- 从$b[m, n]$开始根据箭头上溯至i=0或j=0即可
  - 当$b[i, j]$= “↖”时，有$x_i=y_j$是LCS的一个元素
  - 逆序构造出LCS，可用递归算法顺序打印

```
PRINT_LCS(b, X, i, j)
if i = 0 or j = 0
    return
if b[i, j] = “↖”
    PRINT_LCS(b, X, i-1, j-1)
    print x_i
elseif b[i, j] = “↑”
    PRINT_LCS(b, X, i-1, j)
else PRINT_LCS(b, X, i, j-1)
```

$O(m+n)$

每次递归调用

i和j至少一个会减少1

最长公共子序列LCS (续)

■算法改进

时空性能常数因子改变，运行时间渐近性能不变

- 可去掉表b——提升空间常数因子
  $c[i, j]$只依赖于$c[i-1, j-1]$, $c[i-1, j]$, $c[i, j-1]$
  可在$O(1)$判定是由哪项计算得到的，即可不依赖于b
  在$O(m+n)$时间重构LCS
- c只需要两行——提升空间性能
  c只需要当前行和前一行
  但无法构造出解，空间由$O(mn)$变为$O(n)$

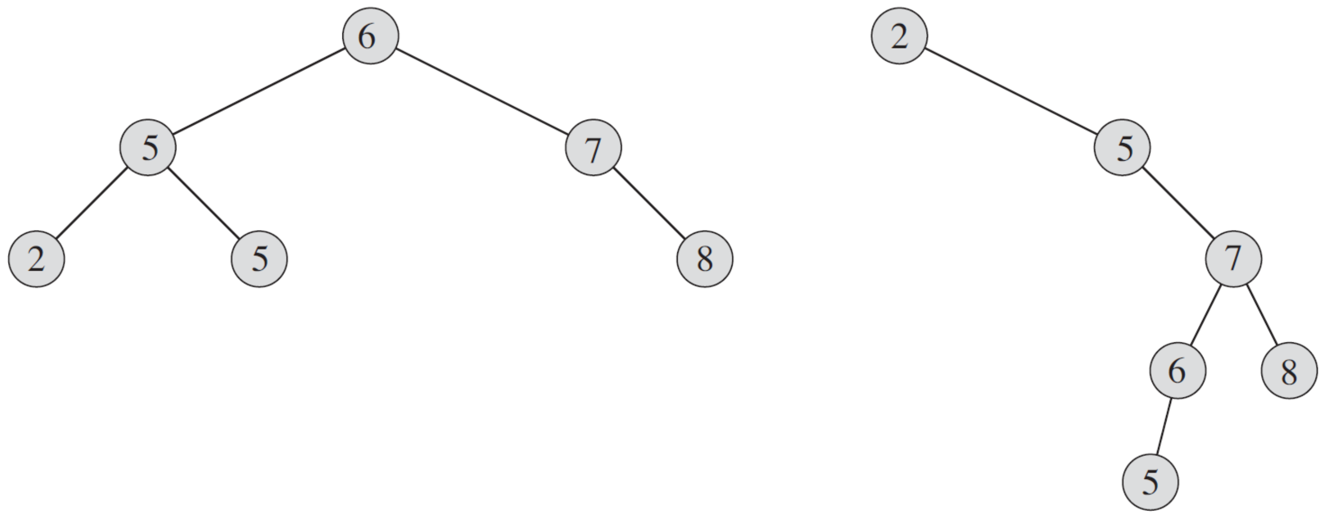
动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- **最优二叉搜索树**
- 0-1背包

最优二叉搜索树

- 二叉搜索树（教材p161）：设x是二叉搜索树中的一个结点，如果y是x左子树中的一个结点，那么$y.key \le x.key$；如果y是x右子树中的一个结点，那么$y.key \ge x.key$
  - 若它的左子树不空，则左子树上所有结点的值均小于等于它的根结点的值
  - 若它的右子树不空，则右子树上所有结点的值均大于等于它的根结点的值
  - 它的左、右子树也分别为二叉搜索树

最优二叉搜索树 (续)



(a) (b)

图 12-1 二叉搜索树。对任何结点 x，其左子树中的关键字最大不超过 $x.key$，其右子树中的关键字最小不低于 $x.key$。不同的二叉搜索树可以代表同一组值的集合。大部分搜索树操作的最坏运行时间与树的高度成正比。(a)一棵包含 6 个结点、高度为 2 的二叉搜索树。(b)一棵包含相同关键字、高度为 4 的低效二叉搜索树

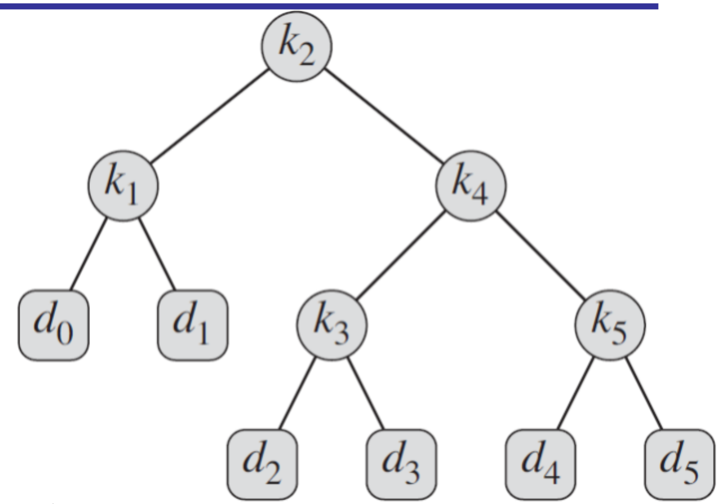
最优二叉搜索树 (续)

- 二叉搜索树上的基本操作花费时间与树高成正比
  - n个结点的完全二叉树：最坏运行时间$\Theta(\lg n)$
  - n个结点的线性链：最坏运行时间$\Theta(n)$
  - 随机构造的二叉搜索树期望高度为$O(\lg n)$，基本操作的平均运行时间$\Theta(\lg n)$

最优二叉搜索树 (续)

- **最优二叉搜索树**：给定一个n个不同关键字的已排序的序列$K=<k_1, k_2, ..., k_n>$ $(k_1<k_2<...<k_n)$，构造一棵**期望搜索代价最小**的二叉搜索树
  - 每个关键字k_i：以概率p_i搜索
  - 搜索的值不在K中：$n+1$个伪关键字d_i $(i=0, 1, ..., n)$，分别以q_i概率搜索
    - d_0：所有小于k_1的值
    - d_n：所有大于k_n的值
    - d_i $(i=1, 2, ..., n\text{-}1)$：所有大于$k_i$小于$k_{i+1}$的值

最优二叉搜索树 (续)



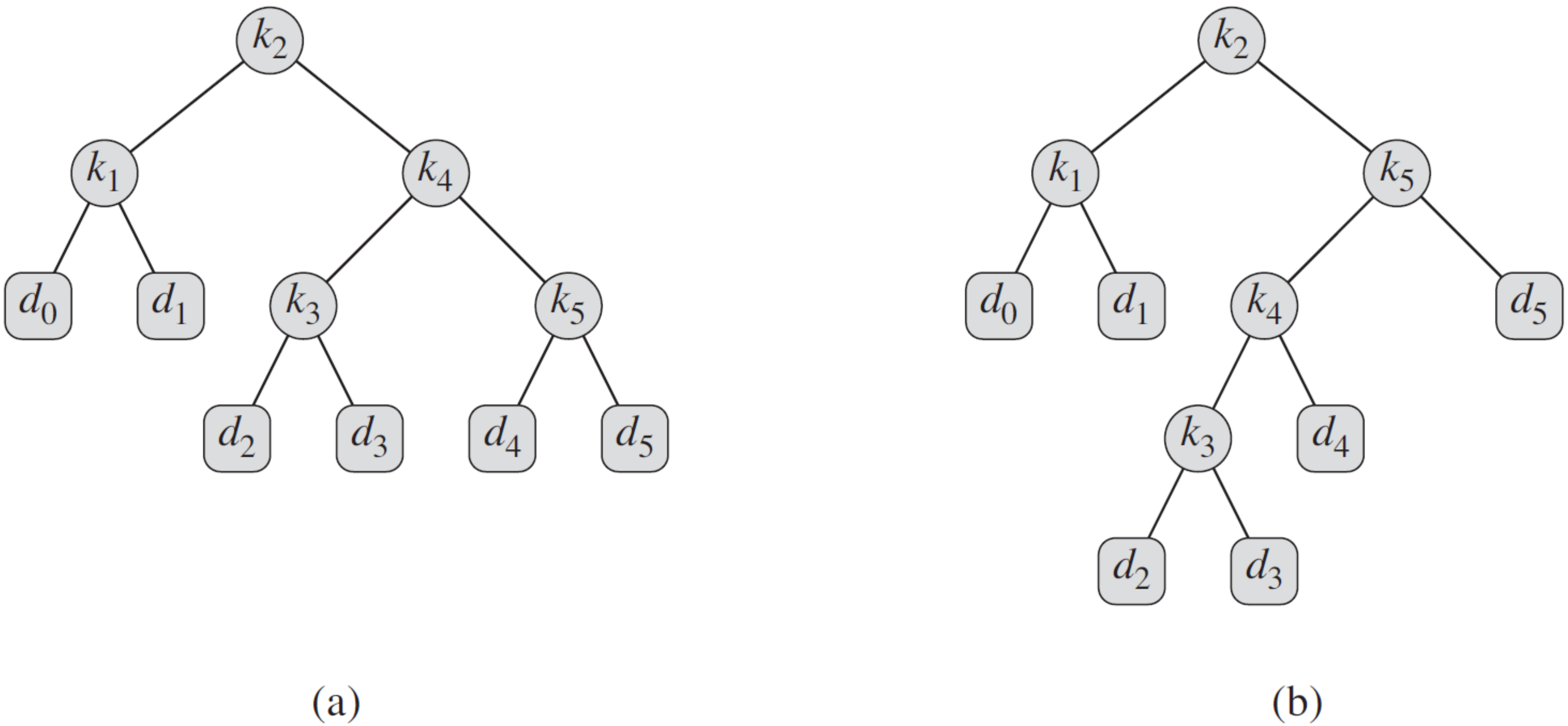
■要么搜索成功要么搜索失败：

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

■二叉搜索树进行一次搜索的期望代价：

$$\begin{aligned} \mathrm{E}[\text{search cost in T}] &= \sum_{i=1}^{n} (\mathrm{depth}_{\mathrm{T}}(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_{\mathrm{T}}(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_{\mathrm{T}}(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_{\mathrm{T}}(d_i) \cdot q_i \end{aligned}$$

最优二叉搜索树 (续)



i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

期望搜索代价：
(a)2.80
(b)2.75（最优）

最优二叉搜索树 (续)

- 最优二叉搜索树：
  - 不一定高度最矮
  - 概率最高的关键字不一定出现在根结点
- n个结点的二叉树数量为$\Omega(4^n/n^{3/2})$
- 穷举法需要检查指数棵二叉树
- 动态规划求解：
  1. 最优二叉搜索树的结构
  2. 递归算法
  3. 计算最优二叉搜索树的期望搜索代价
  4. 构造最优二叉搜索树

最优二叉搜索树 (续)

1. 最优二叉搜索树的结构

- 考虑一棵二叉搜索树的任意子树T’：
  包含连续关键字$k_i, \ldots, k_j$, $1 \le i \le j \le n$
  叶结点是伪关键字$d_{i-1}, \ldots, d_j$
- 若T’是最优二叉搜索树T的子树，则T’也是最优的
  “剪切-粘贴” + 反证法

- 是否完全定义了子问题空间？—— “空子树”
  $i \ge 1$，$i-1 \le j \le n$
  $j=i-1$时，子树不包含实际关键字，只包含伪关键字d_{i-1}

最优二叉搜索树 (续)

2. 递归算法

- 子问题：求解包含关键字$k_i, \ldots, k_j$的最优二叉搜索树，其中$i \geq 1$且$i-1 \leq j \leq n$
- $e[i, j]$：该最优二叉搜索树中进行一次搜索的期望代价
  - $j=i-1$：子树只包含伪关键字d_{i-1}
    期望搜索代价$e[i, i-1]=q_{i-1}$
  - $j \geq i$：需从$k_i, \ldots, k_j$中选择一个根结点k_r，
    构造包含$k_i, \ldots, k_{r-1}$的最优二叉搜索树作为左子树
    构造包含$k_{r+1}, \ldots, k_j$的最优二叉搜索树作为右子树

最优二叉搜索树 (续)

2. 递归算法

- $e[i, j]$：该最优二叉搜索树中进行一次搜索的期望代价
  - 子树所增加的期望搜索代价：所有概率之和

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

  - 若k_r为$k_i, \ldots, k_j$最优二叉搜索树的根结点，则有

$$e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r+1,j))$$

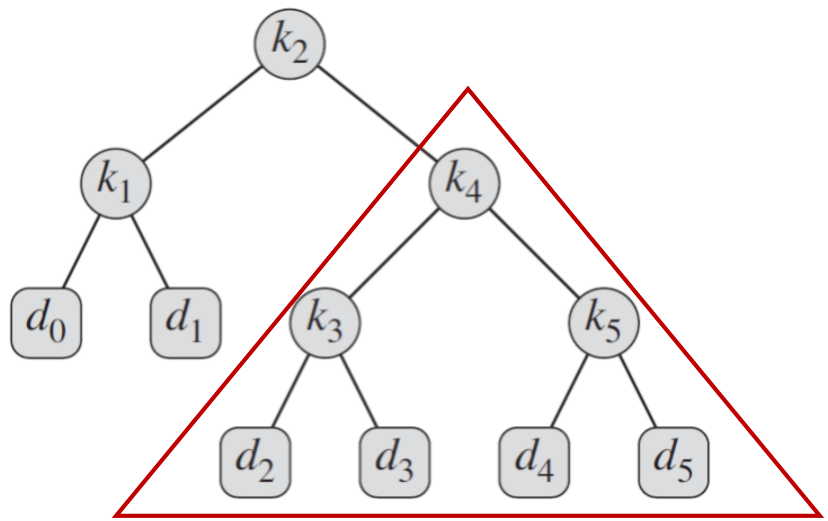
注意到 $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$

因此 $e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$

最优二叉搜索树 (续)

2. 递归算法

➢例：



$$e[1,5] = p_2 + \left(e[1,1] + \underbrace{\sum_{l=1}^{1} p_l + \sum_{l=0}^{1} q_l}_{w(1,1)} \right) + \left(\boxed{e[3,5]} + \underbrace{\sum_{l=3}^{5} p_l + \sum_{l=2}^{5} q_l}_{w(3,5)} \right)$$

$$= e[1,1] + e[3,5] + w(1,5)$$ 其中 $p_2 + w(1,1) + w(3,5) = w(1,5)$

最优二叉搜索树 (续)

2. 递归算法

- 最终递归公式：

$$e[i,j]=\begin{cases} q_{i-1}, & j=i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w(i,j)\}, & i\le j \end{cases}$$

3. 计算最优二叉搜索树的期望搜索代价

- $root[i, j]$：包含$k_i, \ldots, k_j$ ($1\le i\le j\le n$) 的最优二叉搜索树根结点k_r的下标r
- 保存$w[i, j]$的值避免重复计算：

$$w[i,j]=\begin{cases} q_{i-1}, & j=i-1, \\ w[i,j-1]+p_j+q_j, & i\le j \end{cases}$$

最优二叉搜索树 (续)

```
OPTIMAL_BST(p, q, n)
    let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
    for i ← 1 to n+1 do
        e[i, i-1] ← q_{i-1}
        w[i, i-1] ← q_{i-1}
    for l ← 1 to n do
        for i ← 1 to n-l+1 do
            j ← i + l - 1
            e[i, j] ← ∞
            w[i, j] ← w[i, j-1] + p_j + q_j
            for r ← i to j do
                t ← e[i, r-1] + e[r+1, j] + w[i, j]
                if t < e[i, j]
                    e[i, j] ← t
                    root[i, j] ← r
    return e and root
```

$\Theta(n^3)$

最优二叉搜索树 (续)

3. 计算最优二叉搜索树的期望搜索代价——算法改进

   - 教材p231练习15.5-4：对所有$1 \le i < j \le n$，存在最优二叉搜索树，其根满足$root[i, j-1] \le root[i, j] \le root[i+1, j]$
   - 运行时间减少为$\Theta(n^2)$

最优二叉搜索树 (续)

```
OPTIMAL_BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
for i ← 1 to n+1 do
    e[i, i-1] ← q_{i-1}
    w[i, i-1] ← q_{i-1}
for l ← 1 to n do
    for i ← 1 to n-l+1 do
        j ← i + l - 1
        e[i, j] ← ∞
        w[i, j] ← w[i, j-1] + p_j + q_j
        for r ← root[i, j-1] to root[i+1, j] do
            t ← e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] ← t
                root[i, j] ← r
return e and root
```

$\Theta(n^2)$

最优二叉搜索树 (续)

4. 构造最优二叉搜索树

- 教材p230-231练习15.5-1（先序遍历输出BST）

```
CONSTRUCT_OPTIMAL_BST(root)
    return RECURSIVE_CONSTRUCT_OPTIMAL_BST(root, 1, n, 0)
```

```
RECURSIVE_CONSTRUCT_OPTIMAL_BST(root, i, j, indicator)
    if i ≤ j
        if indicator = 0
            print "k"_root[i, j] "为根"
        elseif indicator = 1
            print "k"_root[i, j] "为k"_j+1 "的左孩子"
        else print "k"_root[i, j] "为k"_i-1 "的右孩子"
        RECURSIVE_CONSTRUCT_OPTIMAL_BST(root, i, root[i, j]-1, 1)
        RECURSIVE_CONSTRUCT_OPTIMAL_BST(root, root[i, j]+1, j, 2)
    else
        if indicator = 1
            print "d"_j "为k"_j+1 "的左孩子"
        else print "d"_j "为k"_i-1 "的右孩子"
```